(12) UK Patent Application (19) GB (11) 2 370 894 (13) A

(43) Date of A Publication 10.07.2002

(21) Application No 0125091.9

(22) Date of Filing 18.10.2001

(30) Priority Data
(31) 09695207 (32) 24.10.2000 (33) US

(51) INT CL^7
G06F 9/445

(52) UK CL (Edition T)
G4A AFL

(56) Documents Cited
GB 2227584 A
EP 1058187 A1
US 5701492 A
EP 1087294 A2
US 6052803 A

(58) Field of Search
UK CL (Edition T) G4A AFL
INT CL^7 G06F 9/445
Online : WPI,EPODOC,PAJ,INSPEC,ELSEVIER,TDB

(71) Applicant(s)
Hewlett-Packard Company
(Incorporated in USA - Delaware)
3000 Hanover Street, Palo Alto, California 94304,
United States of America

(72) Inventor(s)
John L Boldon

(74) Agent and/or Address for Service
Carpmaels & Ransford
43 Bloomsbury Square, LONDON, WC1A 2RA,
United Kingdom

(54) Abstract Title

Updating firmware during operation

(57) A laser printer (20) stores a compressed source copy of its operating system (OS1) in flash memory (25). Upon boot-up, the source copy is used to generate an active copy of the operating system that executes from RAM (23) for faster operation. A network management station (13) can transmit an upgrade operating system in compressed form over a network (AP1) to the printer, which treats the upgrade as another print job and places it in its job queue (30) in RAM. When the upgrade reaches the front of the queue, it is written to flash memory, overwriting the source copy of the active operating system. The active copy continues to process print jobs and handle network requests. When an idle state is detected, the active operating system resets the printer, purging itself from RAM. On re-initialization, the recently written source copy of the upgrade operating system is decompressed into RAM and becomes the new active operating system. A spare compressed copy (OS1") of an operating system is maintained in flash memory in case the upgrade procedure fails. When an upgrade failure is detected, the spare operating system is used to generate the active operating system, pending another attempt at the upgrade procedure. The invention applies to application program upgrades as well as operating system upgrades.

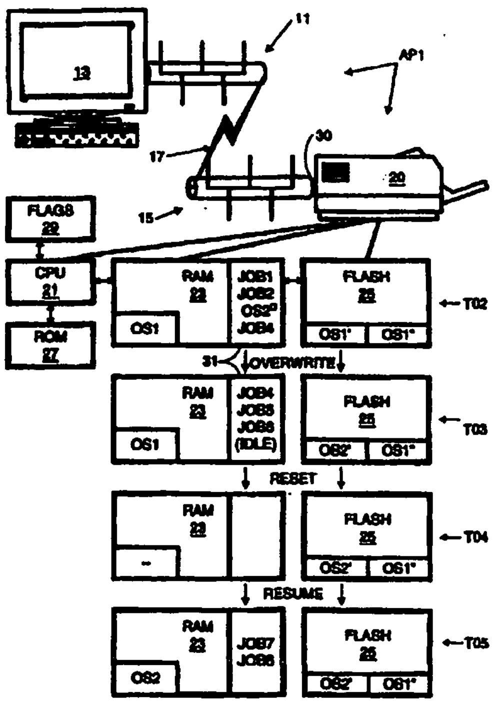
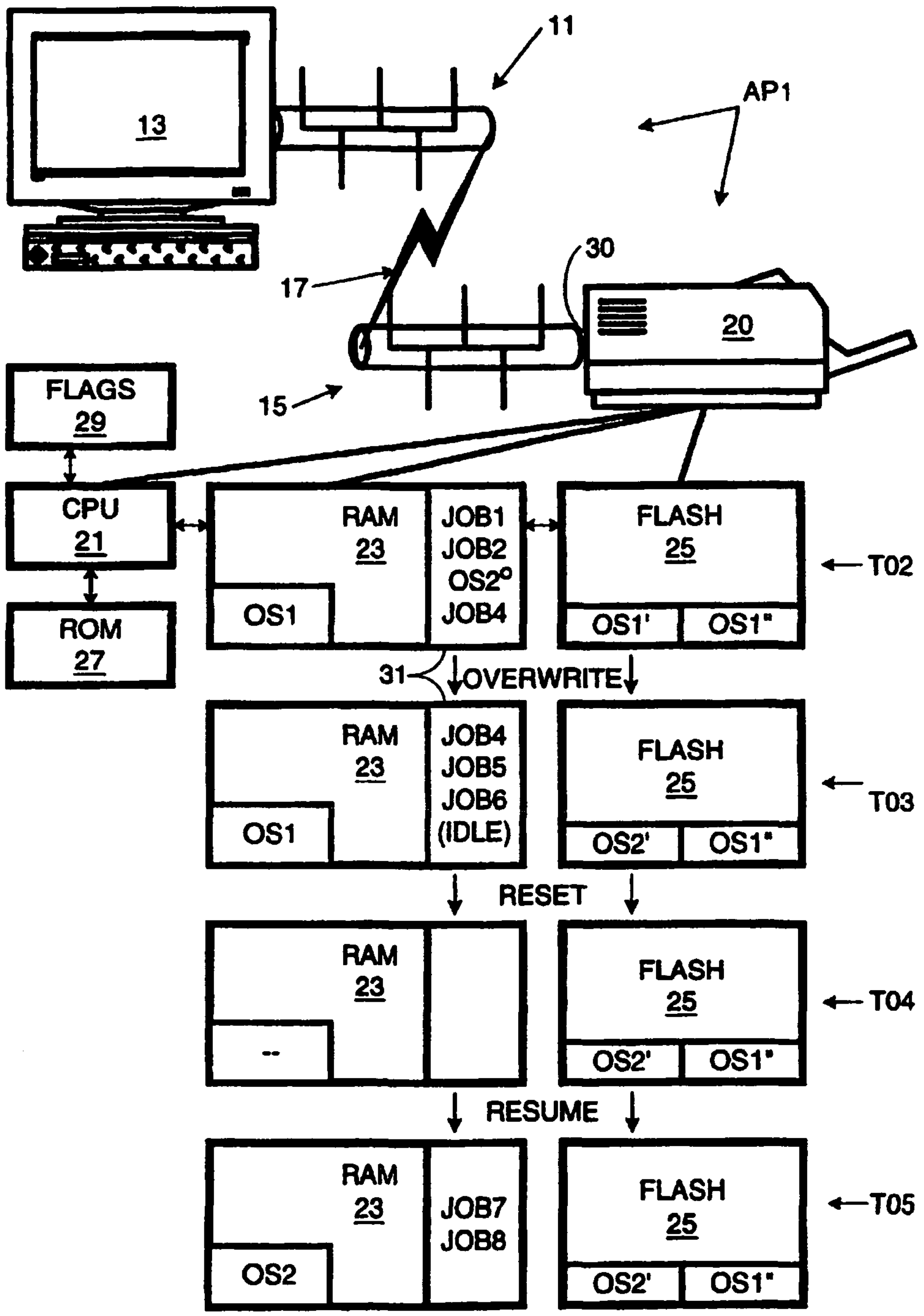


FIG. 1

GB 2 370 894 A



11
AP1
13
17
30
20
15
FLAGS 29
CPU 21
ROM 27
RAM 23
OS1
JOB1
JOB2
OS2°
JOB4
FLASH 25
OS1'
OS1"
T02
31
OVERWRITE
RAM 23
OS1
JOB4
JOB5
JOB6
(IDLE)
FLASH 25
OS2'
OS1"
T03
RESET
RAM 23
--
FLASH 25
OS2'
OS1"
T04
RESUME
RAM 23
OS2
JOB7
JOB8
FLASH 25
OS2'
OS1"
T05

FIG. 1

2370894

Workflow-Friendly Firmware Upgrades for Network Devices

BACKGROUND OF THE INVENTION

The present invention relates to computers and other network devices and, more particularly, to a system for upgrading firmware for network devices, such as printers, modems, and hubs. A major objective of the invention is to provide for workflow friendly firmware upgrades of network devices.

Much of modern progress is associated with the increasing prevalence of computers. Every computer includes at least one processing unit and memory. The processing unit manipulates data in accordance with a program of instructions; the memory stores the instructions and data.

General-purpose computers are designed to accept new programs readily to implement new functions and for convenient updating of old application programs and an underlying operating system program. In addition to general-purpose computers, there is a wide variety of application-specific computer devices, i.e., "embedded devices". Some of these, such as dedicated word processors, are basically computers with fixed programs. Others embedded devices, such as cameras and microwaves, are computer-enhanced versions of traditionally non-computer products. Many computer peripherals, such keyboards and monitors, used to facilitate human-computer interaction, themselves have embedded computers. In addition, there are network peripherals, such as hubs and modems, that enable computers and computing devices to communicate with each other.

In view of their application-specific functions, embedded devices tend to have simpler operating systems and fewer application programs than general-purpose computers do. For many embedded devices, it is feasible and desirable to store operating systems and application programs in non-volatile solid-state memory. In fact, many embedded devices store their operating systems and application programs in solid-state non-volatile memory. In contrast, most general-purpose computers store their operating system and application programs on capacious hard disks; only the BIOS ("basic input-output system) is stored in solid-state non-volatile memory. Programs and data built into a computer device and stored in non-volatile solid-state memory are known as "firmware".

Firmware can be stored in read-only memory (ROM). The firmware programs can be executed directly from ROM. Alternatively, programs can be transferred to volatile memory, e.g., DRAM (dynamic random access memory) for execution. The latter approach requires more DRAM, but achieves higher execution speeds. The choice of ROM versus DRAM-based program execution is often based on a tradeoff between cost and performance.

Since ROM contents cannot be changed, correcting defects or enhancing ROM-based programs often requires physically replacing ROM, which can be inconvenient and expensive. To avoid physically replacing memory, many computing devices store their firmware in re-writable non-volatile solid-state memory, such as "flash" memory. Computing devices that use flash memory for storing updatable programs typically have a media port or a network port for receiving an upgrade. The media port can be a drive or slot for removable media. The network port can provide for digital communications between networked devices without the physical movement of storage media.

"Network devices" are computer devices with network ports. Networks range in complexity from "direct" networks, e.g., a computer in communication with a printer via a parallel port connection, to local-area networks (LANs), to wide-area networks (WANs) to the Internet, which comprises millions of interconnected computing devices. Networked computing devices are often centrally managed over the network for efficiency and to maintain uniformity across the corporation.

Upgrades can be centrally managed by transmitting upgrade files from a network-management station to target network devices. Typically, the upgrades are effected during a "maintenance cycle" in which users are precluded from using the network. Of course, a maintenance cycle interferes with workflow and user productivity, and thus is costly. The cost becomes an even greater concern when upgrades must be implemented separately for each different network device type and model.

In the case of a general-purpose computer with a large hard disk, it is possible to load an upgrade program to an unused area of the hard disk without deleting existing programs. Thus, the existing programs, including the operating system, can resume functioning once the upgrade is installed on the hard disk. However, there is rarely much unused capacity in firmware-based devices, so storing an upgrade program usually requires overwriting the existing firmware.

Since the overwriting effectively destroys the prior firmware, care is typically taken to ensure that no normal jobs are being processed during the upgrade, e.g., printers do not print, scanners do not scan, and modems do not communicate. A minimal special-purpose program (or an external tool) can manage the overwriting, while normal functioning is halted. Once the new firmware is in place, the network device can be rebooted if necessary, and normal and hopefully enhanced device operation can proceed.

In the case of a direct network or a small LAN, it is feasible to allow users to continue work on a computer workstation while a network appliance, such as a printer, is being upgraded. Users can be warned of the upgrade so that they do not assume print jobs are being processed and are not surprised when error messages are generated.

While this approach to upgrades works well for network devices that are dedicated to a host computer or are part of a small LAN, it is not well suited for large LANs and for centralized upgrading of a wide-area network. For example, a management information specialist may be required to upgrade a printer at a remote site. It may not be practical to inform everyone that might want to use the printer that it is being taken off line. In addition, it may be undesirable to interrupt network access to the target device. In some cases, scheduling an upgrade for "off hours" can minimize this problem. However, this can be inconvenient for the person managing the upgrade; also, some companies do not really have "off hours".

If an operating system upgrade fails, the results can be catastrophic for the peripheral. In some cases, e.g., the upgrade process is interrupted by a power failure, the device can be crippled and left non-functional or minimally functional. Inconvenient and expensive on-site repair may be required to resume functionality.

What is needed is a convenient and reliable system for upgrading embedded network devices that interferes minimally, if at all, with normal usage. Preferably, the system would be robust enough to avoid expensive repairs in the event of an upgrade failure.

SUMMARY OF THE INVENTION

The present invention provides for a network device that can upgrade its firmware while maintaining full functional operation. A source copy of program (e.g., operating system or application program) for the network device is stored in re-writable non-volatile memory. During initialization, the source copy is used to generate an active copy of the program that executes from volatile memory. The active program can partially or completely overwrite its source copy with an upgrade program, without requiring a contemporaneous loss of availability of the network device to the network. Instead, the active program can delay re-initialization until a more convenient time, e.g., when an idle state is detected. Upon re-initialization, the active program is purged, and a new active program is generated from the source program resulting from the upgrade.

In practice, a firmware upgrade can be received via a network port (e.g., ethernet, serial port, parallel port). The upgrade, which can be a compressed version of all or part of a program, can be stored in volatile memory pending other events and activities in the device. For a printer, this might include other print jobs ahead of the upgrade job in the print queue. When the upgrade is processed, the upgrade is written to non-volatile memory to yield a source copy of an upgraded program. The active program can handle functions performed by the network device before or after the writing to non-volatile memory. If the active program supports parallel functions, these may also be handled normally during the actual writing of the upgrade to nonvolatile memory. In addition, pipelined activities can continue during the firmware overwriting. The upgrade can be completed by re-initializing the network device, purging the prior active program and replacing it with an active copy of the upgrade program.

Preferably, the time for this re-initialization can be determined by the prior active program in response to a detection of a predetermined state, e.g., an idle or a safe state. The re-initialization can be triggered, for example, by a soft reset of the network device. Alternatively, the invention provides for a switch to a new program without a reset.

The present invention requires two copies of the program being replaced, whereas network devices that execute programs from non-volatile memory require only one. However, since only one copy is used for operating the network device, the other can be compressed, reducing the amount of additional memory required. On the other hand, executing the active copy of the program in volatile memory, which tends to be faster than non-volatile memory, can provide for a higher performance network device relative to network devices that execute the program from non-volatile memory.

The invention provides for storing a "spare" source copy of a program in non-volatile memory. An initialization routine can check for the integrity of the main source program; if the main source fails the check, the initialization routine can generate an active program from the spare. The spare can be stored in read-only memory or in rewritable non-volatile memory. The spare can be a copy of the current or a prior source program. Alternatively, the spare can be a skeletal program intended solely for reloading a full-featured program in the event of an upgrade failure. If an upgrade failure is detected, the initialization routine can activate the spare program. Depending on the implementation of the spare, it can be used directly from non-volatile memory, or transferred (and decompressed if compressed) to volatile memory for execution. The spare allows a failed upgrade to be repeated or corrected without expensive and inconvenient local repair.

Accordingly, the present invention allows the program of a network device to be upgraded with minimal impairment of its availability to users. Potential users do not need to be warned of the upgrade procedure. Queued jobs do not need to be purged. The device does not even have to stop receiving new work either after the upgrade has been received or after the nonvolatile memory has already been rewritten. Re-initialization can be delayed until a convenient time so that device shutdown is not required during a period of high demand. A spare non-volatile copy of a program can be used to allow remotely managed recovery from failed upgrade procedures. These and other features and advantages of the invention are apparent from the description below with reference to the following drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a schematic diagram of a network including a printer with an upgrade system in accordance with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In accordance with the present invention, a network system AP1 comprises a management LAN 11 with a network management station 13, and a remote LAN 15. LAN 11 and LAN 15 are coupled by a satellite link 17. Remote LAN 15 includes a target printer 20, that includes a central processing unit 21, volatile RAM 23, and non-volatile flash 25, read-only memory 27, and a flag register 29. Printer 20 is coupled to the rest of remote LAN 15, and thus to network system AP1 via its network port 30.

The contents of memories 23 and 25 change over time. Normally, flash 25 stores a source operating system in compressed form, plus a spare copy of the source operating system, also in compressed form. RAM 23 stores the active operating system for execution by CPU 21 while printer 20 is on. ("Operating system" is being used in the broad sense, including all programs used to effect normal functioning of printer 20.) During an upgrade procedure, a compressed version of an upgrade operating system can be stored in RAM 23 temporarily pending transfer to flash 25. The various storage permutations are described in greater detail below with reference to time-tables I, II, and III.

Time-Table I: First Upgrade Example				
Time	Active OS	Upgrade OS	Source OS	Spare OS
T00	--	--	OS1'	OS1"
T01	OS1	--	OS1'	OS1"
T02	OS1	OS2°	OS1'	OS1"
T03	OS1	--	OS2'	OS1"
T04	--	--	OS2'	OS1"
T05	OS2	--	OS2'	OS1"

Time T00 corresponds to the state of printer 20 when it is first installed and prior to being powered on. A compressed copy OS1' of the main operating system is stored in flash 25, which also stores a spare copy OS1" of the original operating system. Nothing is stored in RAM 23 in the power-off state.

During boot-up, a boot sequence, stored in ROM 27 accesses flash 25 and decompresses operating system copy OS1' to generate active operating system copy OS1 in RAM 23. Time T01 corresponds to a normal operating condition after boot-up and prior to any upgrades. Two copies OS1' and OS1" of the main operating system OS1 continue to be stored in flash 25. Operating system OS1 manages print jobs, as well as other communications such as status requested, received from host computers, typically on remote LAN 15. Two of these jobs, JOB1 and JOB2, remain in a print-job queue 31 when management station 13 initiates an upgrade procedure directed at printer 20.

Management station 13 begins the upgrade procedure by transmitting a copy OS2' of an upgrade operating system in compressed form. Compressed operating system copy OS2' is presented in the format of a print-job file to printer 20 so that it appears as just another print job that is inserted in queue 31 behind pending jobs JOB1 and JOB2. While the upgrade job is pending, other print jobs, e.g., JOB4, can be received. This is the situation depicted at time T02 in FIG. 1 as well as in time-table I.

In contrast to the prior art, the upgrade procedure does not require other jobs to be halted. Instead, print jobs JOB1 and JOB2 are processed in the order they were received. When printer 20 processes the upgrade job, it overwrites the main source operating system copy OS1' in flash with the compressed copy OS2' of the upgrade operating system, as indicated in FIG. 1 and time-table I at time T03. Job JOB2 may continue during the operating system overwrite, and even after overwriting is completed. In addition, printer 20 can respond to status and other requests from other network devices. For example, a workstation may be checking the printer's internal web page for status and may be signing up for e-mail alerts. Job JOB4 remains pending, along with other jobs JOB5 and JOB6 received during or after the upgrade job is executed.

Eventually, printer 20 detects an idle period IDLE, indicated in FIG. 1, with no jobs and no inquiries (as to printer status) pending. Printer 20 resets itself at time T04. In the course of the reset, operating system copy OS1 and any other contents of RAM 23 are purged, as indicated in FIG. 1 and time-table I.

In response to the reset, printer 20 re-boots. The boot code in ROM 27 instructs printer 20 to decompress the main operating system copy in flash 21, which is now OS2'. Thus, an uncompressed version of OS2 is stored in RAM 23, for execution by CPU 21, as shown for time T05 in FIG. 1 and time-table I. Once again, printer 20 is accepting print jobs, e.g., JOB7 and JOB8 for processing. This normal mode of operation can continue until an idle state (IDLE) is detected.

Time T11 in time-table II below corresponds to a time well past T05 in the same basic state of operation, but just before a second upgrade procedure. Times T12-T15 in the second upgrade procedure correspond generally with times T02-T05 in the first upgrade procedure. However, instead of operating system copy OS2' overwriting operating system copy OS1', operating system copy OS3' overwrites operating system copy OS2'. The principal difference between the second and first upgrades is that in the former case the spare operating system copy OS1" does not match either the main operating system copy OS2' in flash or the upgrade operating system OS3'.

Time-Table II: Second Upgrade Examples				
Time	Active OS	Upgrade OS	Source OS	Spare OS
T11	OS2	--	OS2'	OS1"
T12	OS2	OS3°	OS2'	OS1"
T13	OS2	--	OS3'	OS1"
T14	--	--	OS3'	OS1"
T15	OS3	--	OS3'	OS1"

The purpose of spare operating system copy OS1" is to handle failed upgrades, such as the one characterized below in time-table III. Time T20 corresponds to a power-off state after operating system copy OS3' has been stored. Nothing is in RAM, while flash 25 holds the main operating system copy OS3' and spare operating system copy OS3". Time T21 corresponds to printer 20 after booting; in this state, operating system OS3 is the active operating system executed by CPU 21 from RAM 23. At time T22, a third upgrade procedure involves transfer of a operating system copy OS4' to RAM 23.

Time-Table III: 3rd Upgrade (Failure) Example				
Time	Active OS	Upgrade OS	Source OS	Spare OS
T20	--	--	OS3'	OS1"
T21	OS3	--	OS3'	OS1"
T22	OS3	OS4°	OS3'	OS1"
T23	OS3	--	X	OS1"
T24	--	--	X	OS1"
T25	OS1	--	X	OS1"
T26	OS1	OS4°	X	OS1"
T27	OS1	--	OS4'	OS1"
T28	--	--	OS4'	OS1"
T29	OS4	--	OS4'	OS1"

In this example, the write overwrite is unsuccessful as indicated by the X in the source OS column at time T23. More specifically, operating system copy OS3' in flash is destroyed by an overwrite operation, but to whatever extent OS4° was written to flash 21, that copy is defective or incomplete. The failure could be due to a defect in upgrade file OS4°, either due to a design error or a transmission error, or due to an overwrite error. The most likely cause of the failure is a loss of power during overwriting, in which case the state represented at time T23 would be skipped.

At time T25, as printer 20 is booting up, the defective character of operating system copy OS4' in flash 25 is detected. This detection can involve reading a flag in flag register 29. When the overwrite process is begun at time T23, a flag in flag register 29 is set that is to be reset when the overwrite is complete. If, upon boot-up, it is determined that the flag has not been reset, for example, due to a power failure during the write process, the boot sequence decompresses spare operating system OS1" instead of the main operating system. Also, if a checksum or other error is detected in the main flash operating system copy OS4', the boot sequence decompresses spare operating system OS1". In either case, the result is shown at time T25, which shows operating system OS1 active in RAM 23.

Optionally, the boot sequence can set an error flag in flag register 29 when the spare operating system is decompressed instead of the main operating system. Active operating system OS1 checks this flag, and, if it is set, sends a warning over the network that can be used by management station 13 to indicate the upgrade procedure failed. In the preferred embodiment, management station 13 can poll printer 20 for its operating system version and determine in that way whether the upgrade was a success or not.

However it recognizes an upgrade failure, management station 13 can send another upgrade operating system copy OS4', which is stored in RAM 23 in compressed form as indicated at time T26. At time T27, operating system OS4' is successfully written to flash 21. During the next idle state, operating system OS1 resets printer 20 at time T28. Upon boot-up, operating system OS4 is running on CPU 21. Note that no local intervention is required despite the failure of the first attempt to install operating system OS4.

While in the illustrated embodiment is a printer, the invention equally applies to other devices such as modems, scanners, facsimile machines, etc. Note that the spare operating system can be in ROM instead of flash. However, maintaining the spare in flash allows it to be upgraded, although precautions must be taken to avoid corrupting the spare.

While the illustrated embodiment deals with operating system upgrades, other types of program upgrades are provided for as well. For example, a flash BIOS can be updated, as well as an application program that includes a facility for overwriting its source copy.

In the illustrated embodiment, it is the operating system that is upgraded. Alternatively, it can be a firmware application program that is upgraded. In the case of a system, like a general-purpose computer, with a firmware BIOS, it can be the BIOS that is upgraded. The invention simply requires a program that overwrites itself. However, there is flexibility in defining what that program is. For example, an operating system according to one conceptualization can comprise the code for an operating system plus and application program for another conceptualization.

The foregoing discussion is directed to a network printer. The invention applies as well to other network devices with programs in solid-state non-volatile memory. Examples of such devices are scanners, modems, digital cameras, digital camcorders, multi-function machines, and network hubs, switches, and routers. In addition, the invention applies to firmware upgrades for general-purpose computers, e.g., BIOS upgrades, and upgrades of firmware for hand-held computers with firmware operating system and application programs.

The present invention has applicability to computer networking and method of upgrading network devices. It is particularly valuable where upgrading is initiated at one location and directed to one or more remote locations. Interference with users is minimized, while convenience to the network manager is enhanced. While the illustrated embodiment has a number of useful features, the scope of the invention is defined by the following claims.

What Is Claimed Is:

CLAIMS

<u>What Is Claimed Is:</u>

1. A method of upgrading firmware for a network device, said network device being designed to perform normal device jobs, said method comprising the steps of:

a) an active copy (OS1) of a first program at least partially overwriting a source copy (OS1') of said first program with a program upgrade (OS2°) to yield a source copy of a second program, said active copy of said first program executing from volatile solid-state memory, said source copies being stored in non-volatile solid-state memory;

b) said active copy of said first program executing a first normal device job (JOB2);

c) initializing said network device so that said active copy of said first program is purged and so that an active copy (OS2) of said second program is generated from said source copy of said second program; and

d) said active copy of said second program executing a second normal device job (JOB7).

2. A method as recited in Claim 1 wherein, after step *d*, said active copy of said first program triggers said initializing in response to a detection of an idle state (IDLE) of said network device.

3. A method as recited in Claim 1 wherein, during step *d*, if a problem with the integrity of said source copy of said second program is detected, an active copy of a program is generated from a spare copy (OS1") of a program.

4. A method as recited in Claim 1 wherein, prior to step a, said source copy of said second program is received by said network device at a network port of said network device.

5. A method as recited in Claim 1 wherein, during step b, said first normal device job is one of plural device jobs in a queue (31) managed by said active copy of said first program, said queue temporarily including said source copy of said second program, step c not occurring until all device jobs ahead of said source copy of said second program in said queue have been at least partially executed.

6. A network device (20) for executing normal device jobs, said network device comprising:

volatile memory (23) for storing active copies of programs including a copy (OS1) of a first active program;

non-volatile memory (25) for storing source copies of said programs, including a source copy (OS1') of a first program;

a network port (30) for receiving network device jobs (JOB1-JOB8) and for receiving an upgrade copy (OS2) of a second program; and

a processor for running said active copy of said first program so that it overwrites said source copy of said first program with said source copy of said second program, then executes some normal device jobs (JOB4-JOB6), then initializes said network device so as to purge said active copy of said first program, said processor generating an active copy (OS2) of said second program from said source copy of said second program, said processor executing said active copy of said second program so as to execute additional normal device jobs (JOB7, JOB8).

7. A network device as recited in Claim 6 wherein said network device has an idle state (IDLE), said active copy of said first program including means for detecting said idle state, said processor initializing said network device in response to a detection of said idle state by said active copy of said first program.

8. A network device as recited in Claim 6 wherein said source copy of said first program and said source copy of said second program are compressed.

9. A network device as recited in Claim 8 further comprising:

a spare copy (OS1") of a spare program stored in said non-volatile memory; and

means (29) for detecting during initialization a defect in said source copy of said second program, said processor, in the event such a defect is detected, generating an active program in said volatile memory from said spare program.

Application No:	GB 0125091.9	**Examiner:**	Nik Dowell
Claims searched:	1 to 9	**Date of search:**	1 May 2002

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.T): G4A (AFL)

Int Cl (Ed.7): G06F (9/445)

Other: Online : WPI,EPODOC,PAJ,INSPEC,ELSEVIER,TDB

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
A	GB 2 227 584 A	(Int Computers) see abstract	-
X	EP 1 087 294 A2	(Nortel Networks) see especially, page 3, line 28 to page 4, line 35	1 and 6 at least
X	EP 1 058 187 A1	(Sagem) see abstract	1 and 6 at least
A	US 6 052 803	(3COM) see abstract	-
X	US 5 701 492	(Canon) see especially column 3, line 20 to column 4, line 5	1 and 6 at least

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		